Nécessité faite loi

Pierre-Marie Pédrot, Alexis Saurin

Nécessité faite loi

Pierre-Marie Pédrot[1] & Alexis Saurin[1]

Laboratoire PPS,
CNRS, UMR 7126, Univ Paris Diderot, Sorbonne Paris Cité,
PiR2, INRIA Paris Rocquencourt, F-75205 Paris, France
{pedrot,saurin}@pps.univ-paris-diderot.fr

Résumé

À partir de la réduction linéaire de tête, nous dérivons de manière systématique un calcul en appel par nécessité. L'introduction d'un calcul pour la réduction linéaire de tête, basée sur une analyse fine de la notion de radicaux premiers de Danos et Regnier, nous permet de construire pas à pas un lambda-calcul en appel par nécessité que l'on compare aux calculs présents dans la littérature.

1. Introduction

Évaluation paresseuse. L'évaluation paresseuse a été proposée par Wadsworth dans sa thèse [18] comme une manière de résoudre une tension entre les stratégies en appel par nom et en appel par valeur. En posant comme il est habituel $\Delta \equiv \lambda x.\,(x)\,x$, $I \equiv \lambda y.\,y$ et en notant $\to_{\mathrm{cbn}}$ (resp. $\to_{\mathrm{cbv}}$) la réduction associée à l'appel par nom (resp. par valeur), on constate aisément que l'appel par valeur est susceptible d'effectuer des calculs inutiles, qui peuvent d'ailleurs conduire à une non-terminaison du calcul (on mettra en évidence le radical impliqué dans une réduction en le représentant de manière grisée) :

$$\begin{array}{rcl} t & \equiv & \boxed{(\lambda x.\,I)\,((\Delta)\,\Delta)} \to_{\mathrm{cbn}} I \\ t & \equiv & (\lambda x.\,I)\,\boxed{((\Delta)\,\Delta)} \to_{\mathrm{cbv}} t \to_{\mathrm{cbv}} \dots \end{array}$$

Ici, l'appel par valeur va s'obstiner à réduire $(\Delta)\,\Delta$ sans s'apercevoir que cet argument ne sert à rien.

On constate tout aussi aisément que l'appel par nom est conduit à effectuer des calculs redondants :

$$\begin{array}{rcl} u & = & \boxed{(\Delta)\,((I)\,I)} \to_{\mathrm{cbn}} \boxed{((I)\,I)}\,((I)\,I) \to_{\mathrm{cbn}} \boxed{(I)\,((I)\,I)} \to_{\mathrm{cbn}} \boxed{(I)\,I} \to_{\mathrm{cbn}} I \\ u & = & (\Delta)\boxed{((I)\,I)} \to_{\mathrm{cbv}} \boxed{(\Delta)\,I} \to_{\mathrm{cbn}} \boxed{(I)\,I} \to_{\mathrm{cbn}} I \end{array}$$

L'appel par nom va ici dupliquer le calcul de $(I)\,I$ alors que l'appel par valeur, ayant fait le calcul avant la substitution, va dupliquer une valeur.

L'appel par nécessité consiste à résoudre cette tension en cherchant à n'effectuer un calcul qu'au cas où ce calcul est nécessaire à la progression de l'évaluation et, dans ce cas, à éviter de répéter des calculs inutilement. La solution de Wasdworth consiste à introduire une réduction de graphe qui permet de partager effectivement des sous-termes. L'appel par nécessité est résumé de plusieurs manières dans la littérature. Ainsi, Danvy et al. [10] résume l'appel par nécessité par la formule suivante :

Demand-driven computation & memoization of intermediate results [10]

En ce sens, l'appel par nécessité est une optimisation aussi bien de l'appel par nom que de l'appel par valeur. Il est cependant plus proche de l'appel par nom que de l'appel par valeur car il a la même notion de convergence et en cela l'équivalence observationnelle associée à l'appel par nécessité coïncide avec celle de l'appel par nom.

Le λ-calcul par nécessité d'Ariola et Felleisen. En 1994, deux groupes de chercheurs ont, de manière simultanée et indépendante, proposé des λ-calculs pour l'appel par nécessité, d'un côté Ariola et Felleisen, de l'autre Maraist, Odersky et Wadler. Les deux propositions, soumises à la même conférence, ont donné lieu à une publication conjointe à POPL 1995 [4] mais à deux versions journal distinctes [3, 15], les auteurs faisant des choix de conception différents. Le calcul d'Ariola et Felleisen peut être présenté comme suit :

Définition 1. Le λ-calcul par nécessité d'Ariola et Felleisen est défini par la syntaxe et les règles de réduction suivantes :

Syntaxe

Terme	t, u	$::=$	$x \mid \lambda x.\, t \mid (t)\, u$
Valeur	v	$::=$	$\lambda x.\, t$
Réponses	A	$::=$	$v \mid \lambda x.\, A\, t$
Contexte d'évaluation	E	$::=$	$[\cdot] \mid E\, t \mid (\lambda x.\, E)\, t \mid \lambda x.\, E[x]\, E$

Réductions

(Deref)	$(\lambda x.\, E[x])\, v$	$\to$	$(\lambda x.\, E[v])\, v$
(Lift)	$(\lambda x.\, A)\, t\, u$	$\to$	$(\lambda x.\, (A)\, u)\, t$
(Assoc)	$(\lambda x.\, E[x])\, (\lambda y.\, A)\, t$	$\to$	$(\lambda y.\, (\lambda x.\, E[x])\, A)\, t$

Intuitivement, la syntaxe et les règles précédentes doivent être comprises comme suit :

— C'est la structure des contextes qui est responsable du caractère paresseux de ce calcul : le terme $E[x]$ met en évidence le fait que la variable x est en position nécessaire dans le terme $E[x]$.

— La règle Deref va alors permettre d'aller chercher le contenu d'un argument à condition qu'il soit déjà calculé et que cet argument soit identifié comme nécessaire car il est passé à une variable nécessaire ; dans ce cas, une substitution a lieu, mais linéairement. L'application n'a en effet pas disparu, et seule une occurrence de la variable a été substituée. (E est en effet un contexte à un seul trou.)

— Les deux autres règles Lift et Assoc vont alors autoriser la commutation des contextes d'évaluation afin de permettre à des radicaux β d'apparaître malgré les lieurs qui persistent.

Exemple de réduction dans le calcul d'Ariola et Felleisen.

$$
\begin{array}{lll}
(\Delta)(I)I & \equiv & (\lambda x.\, (x)\, x)\, (\lambda y.\, y)\, I \\
& \to_{\textsc{Deref}} & (\lambda x.\, (x)\, x)\, (\lambda y.\, I)\, I \\
& \to_{\textsc{Assoc}} & (\lambda y.\, (\lambda x.\, (x)\, x)\, I)\, I \\
& \to_{\textsc{Deref}} & (\lambda y.\, (\lambda x.\, (\lambda z.\, z)\, x)\, I)\, I \\
& \to_{\textsc{Deref}} & (\lambda y.\, (\lambda x.\, (\lambda z.\, z)\, I)\, I)\, I \\
& \to_{\textsc{Deref}} & (\lambda y.\, (\lambda x.\, (\lambda z.\, I)\, I)\, I)\, I \\
& \to_{gc}^{\star} & I
\end{array}
$$

où gc désigne une réduction éliminant les β-radicaux rendus inutiles par le fait que la variable qu'ils lient n'est pas utilisée dans le corps de l'abstraction. Cette notion de réduction ne fera pas partie de nos réductions mais permet de faciliter la lecture des valeurs ; on utilisera cette notation dans la suite de l'article.

Remarque 1 (Sur l'utilisation d'une syntaxe avec `let`). Il est standard de présenter les calculs en appel par nécessité en étendant le λ-calcul avec une construction `let` qui exprime le partage : $\mathbf{let}\, x = t\, \mathbf{in}\, u$. Dans ce cas, on rajoute une règle pour introduire les `let` :

$$(\beta) \qquad (\lambda x.\, t)\, u \;\to\; \mathbf{let}\, x = u\, \mathbf{in}\, t$$

En ce sens, la construction `let` n'est rien d'autre que le marquage du fait qu'un β-radical a été rencontré.

Même s'il y a des intérêts à travailler avec un `let` explicite, que ce soit du point de vue de l'appel par nécessité ou des liens logiques avec le calcul des séquents [7], nous faisons le choix dans cet article de rester fidèle à la syntaxe du λ-calcul et cela pour une raison fort simple : la réduction linéaire de tête s'exprime très naturellement dans cette syntaxe et le calcul initial d'Ariola et Felleisen y est aussi défini [1]. Par ailleurs, nous allons adopter une approche macroscopique de l'évaluation, compatible avec la réduction linéaire de tête telle que présentée par Danos et Regnier ; dans ce cadre il n'y aurait pas vraiment de sens à vouloir introduire les `let`.

L'objet de cet article étant de mettre en évidence les liens profonds entre réduction linéaire de tête et évaluation paresseuse, il est donc naturel de s'en tenir à la syntaxe du λ-calcul.

Des appels par nécessité ? Le calcul présenté par Maraist, Odersky et Wadler dans leur version finale de 1998 diffère du calcul ci-dessus par plusieurs aspects, mais les deux calculs partagent la même réduction standard.

Dans la mesure où l'appel par nécessité est une optimisation de l'appel par nom visant à résoudre les limitations mentionnées plus haut sur les duplications de calculs, et que l'équivalence observationnelle est identique à celle induite par l'appel par nom, il peut sembler futile de vouloir comparer les différents appels par nécessité et mettre en évidence un appel par nécessité comme étant plus canonique qu'un autre.

Pourtant, dès que l'on souhaite introduire des opérateurs de contrôle en appel par nécessité, on est obligé de changer de perspective : en effet, en ajoutant des opérateurs de contrôle, non seulement on rend observable la différence entre appel par nom et appel par nécessité, mais en outre on peut distinguer plusieurs variantes de l'appel par nécessité [5]. Dans ce contexte, cela fait sens que de vouloir trouver un appel par nécessité canonique.

Contributions et organisation de l'article Dans cet article, nous montrerons que l'on peut retrouver un calcul connu en partant d'une réduction construite via des considérations héritées de la logique linéaire, définie à la section 2 et en le raffinant par étapes successives :

1. Un appel par nom faible généralisé (section 3.1) ;
2. Une mémoïsation par passage de valeurs (section 3.2) ;
3. Un partage de réductions (section 3.3).

Cette méthodologie trouve sa justification dans le fait que le calcul final se trouve être un calcul en appel par nécessité, ce que nous justifierons par la suite. Cela nous conduit donc au slogan :

Nécessité = Calcul à la demande + Mémoïzation + Partage	(Slogan)

ou, plus précisément, à affirmer que l'appel par nécessité n'est rien d'autre qu'une *(i)* réduction linéaire de tête faible, *(ii)* en appel par valeur *(iii)* effectuant un partage des clôtures.

Pour atteindre ce résultat, nous commencerons par examiner en détail la réduction linéaire de tête ce qui nous conduira à en proposer une formulation nouvelle sous forme de calcul. La reformulation de la réduction linéaire de tête et les transformations qui vont suivre reposent essentiellement sur deux idées :

1. Plus exactement, Ariola et Felleisen considèrent les deux présentations du calcul, les `let` servant essentiellement à montrer la complétude de leur calcul vis-à-vis de l'appel par nom

— se laisser guider par la σ-équivalence (que nous discutons en section 2.2) ;
— mettre en évidence la structure de contexte de clôture qui constitue l'une des notions clés de notre construction.

C'est cette reformulation de la réduction linéaire de tête qui nous met sur la voie des transformations suivantes : on restreindra d'abord la réduction à être une réduction faible. On limitera le calcul de manière à ce qu'il ne substitue que des valeurs. Enfin, on atteindra un calcul en appel par nécessité en ajoutant une règle pour partager les clôtures, il s'agit du calcul de Chang et Felleisen [6] à ceci près qu'on effectue la substitution linéaire avec des abstractions persistantes alors que leur calcul effectue une réduction destructrice. Avant de conclure, on discute des extensions pleinement paresseuses des calculs en appel par nécessité.

2. Réduction linéaire de tête

Dans cette section, nous rappelons la définition de la réduction linéaire de tête, sur laquelle se base notre formulation de l'appel par nécessité.

2.1. Présentation historique

Introduite par Danos et Regnier [8, 9], la réduction linéaire de tête a été formulée à la suite d'observations similaires au sein de différents systèmes calculatoires, comme les réseaux de preuve [11, 16], la machine de Krivine [13] et la sémantique des jeux [12]. Selon eux, la réduction implémentée par ces systèmes en sous-main est la réduction de tête linéaire et non pas la réduction de tête habituelle. En effet, ces deux réductions sont observationellement équivalentes dans le λ-calcul pur, et nécessitent la présence d'effets pour être distinguées. Le même phénomène est à l'œuvre dans la distinction entre appel par nom et appel par nécessité. Un effet collatéral de cette apparente identité est l'inexistence quasi-totale de la notion de réduction de tête linéaire dans la littérature.

Afin de pallier ce manque, et dans un souci d'explication, nous redéfinissons ici les notions afférentes. Dans la suite de cette section, nous travaillerons implicitement à α-conversion près et avec les conventions de Barendregt pour éviter la capture inopinée de variables.

Définition 2 (Épine dorsale, variable hoc)**.** Soit t un λ-terme. L'épine dorsale de t est un ensemble $\rceil\, t$ de λ-termes défini inductivement par :

$$\frac{}{t \in \rceil\, t} \qquad \frac{r \in \rceil\, t}{r \in \rceil\, t\, u} \qquad \frac{r \in \rceil\, t}{r \in \rceil\, \lambda x.\, t}$$

De plus, pour tout terme t, $\rceil\, t$ contient par construction exactement une variable dénotée $\mathrm{hoc}\,(t)$ [2].

L'épine dorsale d'un terme t n'est rien d'autre que l'ensemble des sous-termes gauches de t, et $\mathrm{hoc}(t)$ en est la variable la plus à gauche.

Définition 3 (Lambdas de tête, radicaux premiers)**.** Soit t un λ-terme. On définit mutuellement les lambdas de tête $\lambda_h(t)$ et les radicaux premiers $p(t)$ de t par induction sur t comme suit :

$$\begin{array}{rcl@{\qquad}rcl}
\lambda_h(x) & \equiv & \varepsilon & p(x) & \equiv & \emptyset \\
\lambda_h(\lambda x.\, t) & \equiv & x :: \lambda_h(t) & p(\lambda x.\, t) & \equiv & p(t) \\
\lambda_h(t\, u) & \equiv & \begin{cases} \varepsilon & \text{si } \lambda_h(t) = \varepsilon \\ \ell & \text{si } \lambda_h(t) = x :: \ell \end{cases} & p(t\, u) & \equiv & \begin{cases} p(t) & \text{si } \lambda_h(t) = \varepsilon \\ p(t) \cup \{x \leftarrow u\} & \text{si } \lambda_h(t) = x :: \ell \end{cases}
\end{array}$$

2. Pour *head occurrence*, et par assimilation plaisante avec le *hoc* latin.

Remarque 2. On peut voir les lambdas de tête et les radicaux premiers d'une manière alternative, en considérant des blocs d'applications. On a en effet les égalités suivantes :

$$\lambda_h((\lambda x.\, t)\, u\, \vec{r}) = \lambda_h(t\, \vec{r})$$
$$p((\lambda x.\, t)\, u\, \vec{r}) = \{x \leftarrow u\} \cup p(t\, \vec{r})$$

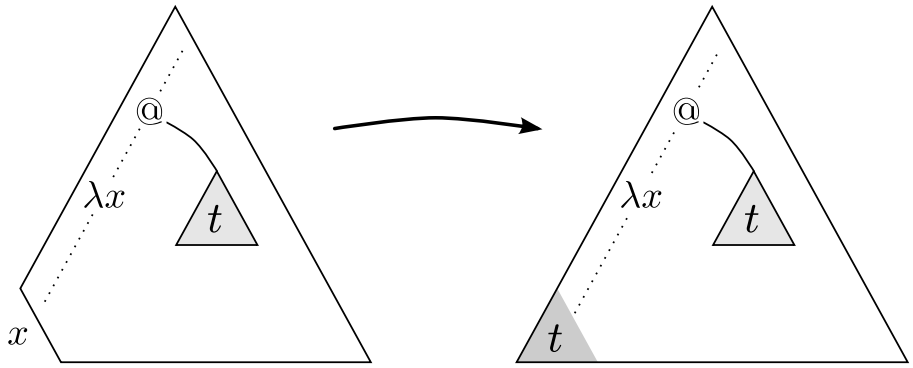
Les lambdas de tête sont les lambdas de l'épine dorsale qui ne recevront pas d'arguments au cours d'une réduction de tête.

Muni de ces notions qui sont illustrées ci-dessous, on peut maintenant définir formellement la réduction linéaire de tête.

Définition 4 (Réduction linéaire de tête)**.** Soit u un λ-terme, soit $x := \mathrm{hoc}\,(u)$. On dit que u se réduit linéairement de tête vers r, noté $u \to_{lh} r$ si :

1. il existe un terme t tel que $\{x \leftarrow t\} \in p(u)$;
2. r est le terme u dans lequel $\mathrm{hoc}\,(u)$, et *uniquement* cette occurrence, a été substituée par t.

Graphiquement :



Remarque 3. La réduction linéaire de tête ne substitue qu'une occurrence d'une variable à la fois, et ne réduit jamais un nœud application, pas plus qu'elle ne diminue le nombre de radicaux premiers : les termes ne font que grossir, d'où le terme de « linéaire » (au sens de *substitution* linéaire).

Voici un exemple de réduction de tête linéaire :

$$\begin{array}{lll}
\Delta\,(I\,I) & \equiv & (\lambda x.\, x\, x)\, ((\lambda y.\, y)\, (\lambda z.\, z)) \\
& \to_{lh} & (\lambda x.\, ((\lambda y_0.\, y_0)\, (\lambda z_0.\, z_0))\, x)\, ((\lambda y.\, y)\, (\lambda z.\, z)) \\
& \to_{lh} & (\lambda x.\, (\lambda y_0.\, \lambda z_1.\, z_1)\, (\lambda z_0.\, z_0)\, x)\, ((\lambda y.\, y)\, (\lambda z.\, z)) \qquad (\star) \\
& \to_{lh} & (\lambda x.\, (\lambda y_0.\, \lambda z_1.\, x)\, (\lambda z_0.\, z_0)\, x)\, ((\lambda y.\, y)\, (\lambda z.\, z)) \\
& \to_{lh} & (\lambda x.\, (\lambda y_0.\, \lambda z_1.\, (\lambda y_2.\, y_2)\, (\lambda z_2.\, z_2))\, (\lambda z_0.\, z_0)\, x)\, ((\lambda y.\, y)\, (\lambda z.\, z)) \\
& \to_{lh} & (\lambda x.\, (\lambda y_0.\, \lambda z_1.\, (\lambda y_2.\, (\lambda z_3.\, z_3))\, (\lambda z_2.\, z_2))\, (\lambda z_0.\, z_0)\, x)\, ((\lambda y.\, y)\, (\lambda z.\, z)) \\
& \to_{gc} & I
\end{array}$$

2.2. Réduction à σ-équivalence près

On remarquera avec profit que la réduction linéaire de tête réduit des termes qui ne sont pas encore des radicaux pour β_h, c'est-à-dire que lh peut aller chercher l'argument d'un lieur qui ne lui

est pas directement appliqué. La troisième réduction de l'exemple de la section précédente, notée ($\star$), en est une illustration. Le lieur λz_1 y saute le radical premier $\{y_0 \leftarrow \lambda z_0.\, z_0\}$ pour aller chercher son argument x. Cette réduction n'aurait pas été possible avec la réduction de tête, β_h.

Ce comportement peut être formalisé plus avant, au travers d'une réécriture à équivalence près, introduite aussi par Regnier [17, 16].

Définition 5 (σ-équivalence)**.** Deux termes t et u sont σ-équivalents, noté $t \cong_\sigma u$, s'ils sont en relation par la clôture des deux générateurs suivants :

$$\begin{array}{rcl} (\lambda x_1.\,(\lambda x_2.\,u)\,t_2)\,t_1 & \cong_\sigma & (\lambda x_2.\,(\lambda x_1.\,u)\,t_1)\,t_2 \\ (\lambda x.\,\lambda y.\,t)\,u & \cong_\sigma & \lambda y.\,(\lambda x.\,t)\,u \end{array}$$

avec les conditions de fraîcheur sur les variables attendues.

La σ-équivalence capture l'intuition que la réduction d'un terme peut faire apparaître des radicaux β qui étaient bloqués par d'autres radicaux essentiellement transparents.

Proposition 1. *Si $t \cong_\sigma u$, alors $p(t) = p(u)$.*

La proposition qui suit souligne le fort lien de parenté qui unit la réduction linéaire de tête avec la σ-équivalence. On rappelle qu'un contexte gauche E est défini par la grammaire inductive suivante :

$$E ::= [\cdot] \mid E\,u \mid \lambda x.\,E$$

Proposition 2. *Si $t \to_{lh} r$ alors il existe deux contextes gauches E_1 et E_2 tels que $t \cong_\sigma E_1[(\lambda x.\,E_2[x])\,u]$ et $r \cong_\sigma E_1[(\lambda x.\,E_2[u])\,u]$.*

En fait, le résultat peut encore être raffiné : la relation $\cong_\sigma$ étant réversible, on peut retrouver r en appliquant à $E_1[(\lambda x.\,E_2[u])\,u]$ les étapes de σ-équivalence inverses de celles appliquées à t pour trouver $E_1[(\lambda x.\,E_2[x])\,u]$. Nous ne détaillerons pas ici précisément cette opération, qui serait peu éclairante et fastidieuse au regard de la présentation qui suit.

2.3. Contextes de clôture

On veut désormais donner un statut de première classe à cette réduction « à σ-équivalence près » en évitant de recourir aux artifices oratoires évoqués juste au-dessus. Pour cela, nous introduisons un contexte de réduction d'un nouveau genre, appelé *contexte de clôture*.

Définition 6 (Contexte de clôture)**.** Les contextes de clôture sont générés par la grammaire inductive suivante :

$$\mathcal{C} ::= [\cdot] \mid (\mathcal{C}_1[\lambda x.\,\mathcal{C}_2])\,t$$

Aussi hétérodoxes que ces contextes puissent paraître, ils possèdent toutes les propriétés requises pour avoir un bon comportement algébrique, à savoir la compositionnalité et la factorisation.

Proposition 3 (Composition)**.** *Si $\mathcal{C}_1$ et $\mathcal{C}_2$ sont deux contextes de clôture, alors $\mathcal{C}_2;\mathcal{C}_1 \equiv \mathcal{C}_1[\mathcal{C}_2]$ est aussi un contexte de clôture.*

Proposition 4 (Factorisation)**.** *Tout terme t peut être uniquement décomposé en un contexte de clôture maximal (au sens habituel de la composition) et un sous-terme t_0.*

Mieux encore, ces contextes capturent précisément la notion de radicaux premiers.

Proposition 5. *Soit t un terme. Alors $\{x \leftarrow u\} \in p(t)$ si et seulement si il existe un contexte gauche E, un contexte de clôture $\mathcal{C}$ et un terme t_0 tels que :*

$$t = E[\mathcal{C}[\lambda x.\,t_0]\,u]$$

Clôtures	c	$::=$	(t, σ)
Environnements	σ	$::=$	$\emptyset \mid \sigma + (x := c)$
Piles	π	$::=$	$\varepsilon \mid c \cdot \pi$
Processus	p	$::=$	$\langle c \mid \pi \rangle$

Push	$\langle (t\,u, \sigma) \mid \pi \rangle$	$\to$	$\langle (t, \sigma) \mid (u, \sigma) \cdot \pi \rangle$
Pop	$\langle (\lambda x.\, t, \sigma) \mid c \cdot \pi \rangle$	$\to$	$\langle (t, \sigma + (x := c)) \mid \pi \rangle$
Grab	$\langle (x, \sigma + (x := c)) \mid \pi \rangle$	$\to$	$\langle c \mid \pi \rangle$
Garbage	$\langle (x, \sigma + (y := c)) \mid \pi \rangle$	$\to$	$\langle (x, \sigma) \mid \pi \rangle$

Figure 1 – Machine de Krivine (avec clôtures)

Moralement, ces contextes sont transparents pour une certaine notion de réduction de tête. On peut en effet voir le contexte $(\mathcal{C}[\lambda x.\, [\cdot]])\, t$ comme un contexte qui ne ferait que d'ajouter la liaison $(x := t)$, et récursivement les liaisons contenues dans $\mathcal{C}$, dans l'environnement.

On peut rendre formelle cette intuition en recourant à la machine de Krivine. On rappelle à la figure 1 la définition de la machine abstraite de Krivine avec clôture (KAM). Comme le montre le résultat suivant, la KAM implémente le calcul des contextes de clôture à l'aide des transitions Push et Pop.

Proposition 6. *Soient t un terme, σ un environnement et π une pile. Pour tout contexte de clôture $\mathcal{C}$, on a :*

$$\langle (\mathcal{C}[t], \sigma) \mid \pi \rangle \longrightarrow^*_{\textsc{Push},\ \textsc{Pop}} \langle (t, \sigma + [\mathcal{C}]_\sigma) \mid \pi \rangle$$

où $[\mathcal{C}]_\sigma$ est définie par induction sur $\mathcal{C}$ comme suit :
- $[[\cdot]]_\sigma \equiv \emptyset$
- $[(\mathcal{C}_1[\lambda x.\, \mathcal{C}_2])\, t]_\sigma \equiv [\mathcal{C}_1]_\sigma + (x := (t, \sigma)) + [\mathcal{C}_2]_{\sigma + [\mathcal{C}_1]_\sigma + (x := (t, \sigma))}$

Réciproquement, pour tous t_0 et σ_0 tels que

$$\langle (t, \sigma) \mid \pi \rangle \longrightarrow^*_{\textsc{Push},\ \textsc{Pop}} \langle (t_0, \sigma_0) \mid \pi \rangle$$

alors il existe $\mathcal{C}_0$ tel que $t = \mathcal{C}_0[t_0]$, où $\mathcal{C}_0$ est défini par induction sur σ_0.

La machine de Krivine fait plus que de calculer les contextes de clôture, grâce aux règles d'accès aux variables Grab et Garbage, qui lui permettent d'implémenter la réduction de tête faible. Mais la proposition précédente souligne bien la similarité de la réduction de tête avec sa cousine linéaire.

En particulier, si l'on désirait fournir une machine abstraite construisant les contextes de clôture, on pourrait simplement adapter la KAM.

2.4. Le calcul λ_{lh}

La présentation historique de la réduction linéaire de tête présente ce défaut qu'est sa définition au travers de structures ad-hoc et peu élégantes, nuisant potentiellement à sa compréhension et à sa manipulation.

En nous appuyant sur le fait que les contextes de clôture capturent les radicaux premiers, on donnera une définition alternative et plus conventionnelle à la réduction linéaire de tête, à base de contextes de réduction. On appellera ce calcul λ_{lh}.

Définition 7 (Calcul λ_{lh})**.** Le calcul λ_{lh} est défini par l'unique règle de réduction

$$E_1[(\mathcal{C}[\lambda x.\, E_2[x]])\, u] \to_{\lambda_{lh}} E_1[(\mathcal{C}[\lambda x.\, E_2[u]])\, u]$$

où E_1, E_2 sont des contextes gauches, $\mathcal{C}$ un contexte de clôture, t et u des termes, et avec les conventions habituelles pour empêcher la capture de variables dans u.

On peut remarquer au passage que cette règle ressemble précisément au calcul du hoc d'un terme et de l'argument correspondant. Dans la règle donnée, x est bien le hoc du terme et, de par le fait qu'on s'autorise de raisonner à contexte de clôture près, on tire de la proposition 5 que $\{x \leftarrow u\}$ est un radical premier. D'où le résultat attendu suivant :

Proposition 7. *Le calcul λ_{lh} capture la réduction linéaire de tête, i.e.*

$$t \to_{\lambda_{lh}} r \qquad ssi \qquad t \to_{lh} r$$

On a donc enfin donné une présentation classique à la réduction linéaire de tête.

3. Vers l'appel par nécessité

Dans la section précédente, on a exprimé la réduction linéaire de tête comme un calcul, λ_{lh}. On va maintenant montrer comment dériver, en trois étapes successives, un λ-calcul en appel par nécessité à partir de ce calcul. On va successivement : *(i)* se limiter à une réduction faible, *(ii)* restreindre la substitution à ne passer que des valeurs et *(iii)* autoriser du partage de contextes de clôture. Au terme de ces trois étapes, on aura obtenu un calcul en appel par nécessité. Plus spécifiquement, on comparera notre calcul à une présentation de l'appel par nécessité introduite récemment et argumenterons que notre approche conduit à une synthèse élégante et canonique.

On verra finalement que la σ-équivalence discutée plus haut nous suggère un peu plus de partage, nous faisant faire une étape de plus vers la pleine paresse [3].

3.1. Réduction linéaire de tête faible

La définition qu'on a donnée de la réduction linéaire de tête au paragraphe 2.4 est une réduction dite *forte*, car elle réduit sous les λ. Ceci n'est pas forcément désirable, pour plusieurs raisons. D'abord, les implémentations « réalistes » du λ-calcul sont toutes en réduction faible, en particulier la KAM. Si l'on veut se baser par la suite sur cette implémentation, mieux vaut nous restreindre de même à une réduction faible. Ensuite, la réduction forte est l'antithèse de l'appel par nécessité : quel est l'intérêt de calculer sous un λ si l'on en a pas encore besoin ? Dans cette optique, mieux vaut présenter un calcul faible.

On peut contraindre la réduction λ_{lh} à ne pas réduire sous les λ en restreignant les contextes d'évaluation sous lequels elle peut agir ; on ne pourrait descendre que sous les contextes de la réduction de tête faible :

$$E^w ::= [\cdot] \mid E^w\, u$$

Cependant, cette contrainte est par trop stricte pour que la réduction puisse continuer à fonctionner correctement. En effet, considérons la règle :

$$E_1^w[\mathcal{C}[\lambda x.\, E_2^w[x]]\, u] \to E_1^w[\mathcal{C}[\lambda x.\, E_2^w[u]]\, u] \qquad (\textsc{Presque})$$

et visualisons-la au travers par exemple de la KAM, qui implémente bien une réduction faible. La décomposition du contexte ne doit se faire que *via* les règles Push et Pop, ce qui est compatible avec la restriction aux E^w. Cependant, il existe des configurations créant des contextes de clôture qui ne relèvent pas de ce motif. Que penser de la situation suivante ?

$$t \quad \equiv \quad \mathcal{C}[\lambda x.\, \lambda z_1. \ldots z_n.\, E^w[x]]\, u\, r_1 \ldots r_n$$

Dans ce cas, les couples de radicaux premiers $\{z_i \leftarrow r_i\}$ sont tous transparents dans la réduction de la KAM et ne participent qu'à la création de la clôture de $E^w[x]$. On a en effet :

$$\langle (t,\sigma) \mid \pi \rangle \longrightarrow^*_{\textsc{Push},\textsc{Pop}} \langle (E^w[x], \sigma + [\mathcal{C}]_\sigma + (x := (u,\sigma)) + (\vec{z} := (\vec{r},\sigma))) \mid \pi \rangle$$

Malgré tout, la règle Presque ne capture pas cette situation, parce que E_2^w interdit la présence de λ-abstractions. En d'autres termes, il faut autoriser la présence de lieurs dans les contextes considérés *sous réserve* qu'ils aient été compensés par suffisamment d'applications antérieures, c'est-à-dire qu'ils participent d'un radical premier.

Les contextes de clôture ne sont pas suffisants pour capturer ce genre de réduction, car il faut pouvoir décomposer le contexte environnant en deux parties. Sur cette base, on pourrait vouloir utiliser des paires de contextes applicatifs et lieurs de la forme suivante :

$$\begin{array}{rcccl} E^{@} & ::= & [\cdot] & \mid & E^{@}\, u \quad (\equiv E^w) \\ E^{\lambda} & ::= & [\cdot] & \mid & \lambda x.\, E^{\lambda} \end{array}$$

Les contextes applicatifs définissent un trop-plein d'applications, et les lieurs consomment le surplus. On ne considérerait alors que les paires de contextes qui, une fois composés, arriveraient à l'équilibre.

Définition 8 (Comblement)**.** Soient $E^{@}$ et E^{λ} des contextes respectivement applicatif et lieur. On dit que $E^{@}$ comble E^{λ} (noté $E^{@} \ni E^{\lambda}$) si $E^{@}$ contient plus d'applications que E^{λ} ne contient de lieurs, ou, formellement :

$$\frac{}{E^{@} \ni [\cdot]} \qquad\qquad \frac{E^{@} \ni E^{\lambda}}{E^{@}\, u \ni \lambda x.\, E^{\lambda}}$$

La règle modifiée deviendrait alors :

$$E^{@}[\mathcal{C}[\lambda x.\, E^{\lambda}[E^w[x]]]\, u] \to E^{@}[\mathcal{C}[\lambda x.\, E^{\lambda}[E^w[u]]]\, u] \qquad \text{si } E^{@} \ni E^{\lambda}.$$

Cela ne suffit pas non plus, car il faut malgré tout être transparent pour les contextes de clôtures. Cependant, cela peut être facilement réparé. Nous introduisons pour cela les contextes applicatifs et lieurs *à clôture près* définis comme suit :

$$\begin{array}{rcccl} \mathcal{C}^{@} & ::= & \mathcal{C} & \mid & \mathcal{C}[\mathcal{C}^{@}\, u] \\ \mathcal{C}^{\lambda} & ::= & \mathcal{C} & \mid & \mathcal{C}[\lambda x.\, \mathcal{C}^{\lambda}] \end{array}$$

D'une certaine façon, ces derniers contextes sont une manière de rendre insensibles les contextes E^{λ} et $E^{@}$ aux contextes de clôture en y introduisant de fines tranches de clôtures. On pourrait aussi l'expliquer sous la forme de systèmes de transition, comme la clôture, à clôtures près [3], des contextes évoqués précédemment : chaque étape Push (respectivement Pop) de la KAM a désormais le droit de faire autant de réductions qu'elle veut sous réserve que l'effet global se résume à faire grossir la clôture du terme courant.

La notion de comblement se relève trivialement à clôture près, et l'on peut finalement définir la réduction linéaire de tête faible. Par facilité de notation, on écrira $\mathcal{C}^w$ pour un contexte applicatif à clôture près que l'on ne désire pas composer avec un contexte lieur [4].

Définition 9 (Réduction λ_{wlh})**.** On définit le calcul de réduction linéaire de tête faible λ_{wlh} par la règle :

$$\mathcal{C}^{@}[(\mathcal{C}[\lambda x.\, \mathcal{C}^{\lambda}[\mathcal{C}^w[x]]])\, u] \to_{\lambda_{wlh}} \mathcal{C}^{@}[(\mathcal{C}[\lambda x.\, \mathcal{C}^{\lambda}[\mathcal{C}^w[u]]])\, u] \qquad \text{si } \mathcal{C}^{@} \ni \mathcal{C}^{\lambda}$$

3. Sans mauvais jeux de mots.
4. Les contextes applicatifs et faibles étant strictement identiques, la notation fait sens.

Remarque 4. Si $\mathcal{C}^{@} \ni \mathcal{C}^{\lambda}$, alors $\mathcal{C}^{@}[\mathcal{C}^{\lambda}]$ est un contexte applicatif, à clôture près.

On peut donner une définition alternative basée sur la version historique de Danos-Regnier, qui hérite des mêmes artéfacts de présentation que la version forte.

Définition 10 (Réduction wlh, version historique)**.** On dit que t se réduit sur r par réduction linéaire de tête faible, noté $t \to_{wlh} r$, si les deux conditions suivantes sont réunies :

1. t se réduit par réduction linéaire de tête sur r ;
2. la liste des lambdas de tête de t est vide.

On voit que cette présentation traite implicitement des contextes lieurs : ne pas avoir de lambdas de tête correspond précisément à avoir ses sous-contextes lieurs comblés. Les deux notions de réduction faible sont alors équivalentes.

Proposition 8. *Le calcul λ_{wlh} et la réduction linéaire de tête faible historique coïncident :*

$$t \to_{\lambda_{wlh}} r \qquad ssi \qquad t \to_{wlh} r$$

3.2. Passage en appel par « valeur »

On définit maintenant une variante en appel par valeur de la réduction linéaire de tête faible, dans un but de diminution des calculs. On ne veut en effet pas devoir dupliquer des calculs déjà faits lors du calcul d'un argument.

Pour ce faire, on va restreindre les contextes provoquant la substitution à ne réagir qu'en présence de valeurs. En outre, dans le même esprit d'insensibilité aux clôtures qui nous a guidés jusqu'ici, nous allons aussi considérer des valeurs à clôture près.

Les valeurs v sont les valeurs habituelles du λ-calcul, c'est-à-dire :

$$v ::= \lambda x.\, t.$$

Les valeurs à clôture près w sont définies directement comme :

$$w ::= \mathcal{C}[v].$$

Pour passer de l'appel par nom à l'appel par valeur, il faut rajouter les contextes forçant les valeurs ; de la même manière, on considère maintenant les contextes qui forcent les valeurs à clôture près . La construction est systématique en appliquant l'encodage habituel de l'appel par valeur.

$$E^{v} ::= [\cdot] \mid E^{v}\, u \mid \mathcal{C}^{@}[(\mathcal{C}[\lambda x.\, \mathcal{C}^{\lambda}[E_1^{v}[x]]])\, E_2^{v}] \mid \mathcal{C}[E^{v}] \qquad \text{où } \mathcal{C}^{@} \ni \mathcal{C}^{\lambda}.$$

Notons que, par souci de lisibilité, on a explicité la possibilité de raisonner à contexte près par une règle dédiée $\mathcal{C}[E^{v}]$, mais on pourrait tout aussi bien procéder au sandwichage en l'*inlinant* dans la définition, comme c'est le cas dans la section précédente.

La réduction linéaire de tête en appel par valeur s'obtient alors directement :

Définition 11. L'appel par valeur linéaire de tête est défini par la réduction :

$$\mathcal{C}^{@}[(\mathcal{C}[\lambda x.\, \mathcal{C}^{\lambda}[E^{v}[x]]])\, w] \to_{\lambda_{wlv}} \mathcal{C}^{@}[(\mathcal{C}[\lambda x.\, \mathcal{C}^{\lambda}[E^{v}[w]]])\, w] \qquad \text{si } \mathcal{C}^{@} \ni \mathcal{C}^{\lambda}$$

On peut remarquer que la règle de réduction n'a pas été beaucoup modifiée ; l'essentiel de la difficulté se trouve dans le choix idoine des contextes.

Ce calcul, quoique désigné sous le nom d'appel par valeur, impémente déjà un appel par nécessité : la réduction a beau ne substituer que des valeurs, elle ne se lance dans le calcul d'un argument que si d'aventure elle a rencontré une variable lui correspondant en position de hoc (adapté à l'appel par valeur). On donne la réduction sur notre exemple au long cours :

$$\begin{array}{lll}
\Delta\,(I\,I) & \equiv & (\lambda x.\, x\,x)\,((\lambda y.\, y)\,(\lambda z.\, z)) \\
& \to_{wlv} & (\lambda x.\, x\,x)\,((\lambda y.\, \lambda z_0.\, z_0)\,(\lambda z.\, z)) \\
& \to_{wlv} & (\lambda x.\, (\lambda y_1.\, \lambda z_1.\, z_1)\,(\lambda z_2.\, z_2)\,x)\,((\lambda y.\, \lambda z_0.\, z_0)\,(\lambda z.\, z)) \\
& \to_{wlv} & (\lambda x.\, (\lambda y_1.\, \lambda z_1.\, x)\,(\lambda z_2.\, z_2)\,x)\,((\lambda y.\, \lambda z_0.\, z_0)\,(\lambda z.\, z)) \\
& \to_{wlv} & (\lambda x.\, (\lambda y_1.\, \lambda z_1.\, ((\lambda y_2.\, \lambda z_3.\, z_3)\,(\lambda z_4.\, z_4)))\,(\lambda z_2.\, z_2)\,x)\,((\lambda y.\, \lambda z_0.\, z_0)\,(\lambda z.\, z)) \\
& \to_{gc} & I
\end{array}$$

On peut bien constater dans la première transition le fait que la réduction va s'opérer dans l'argument qui a été demandé par x, avant de renvoyer une valeur qui sera substituée par la deuxième réduction.

3.3. Partage de clôtures

Le fait de n'autoriser à substituer que des valeurs est un premier pas pour implémenter l'appel par nécessité, mais il n'est pas suffisant pour atteindre les calculs considérés dans la littérature. En effet, si l'on observe bien la règle de réduction de la section précédente, on peut être témoin d'une duplication inutile des calculs :

$$(\mathcal{C}'[\lambda x.\, E^v[x]])\,\mathcal{C}[v] \to_{\lambda_{wlv}} (\mathcal{C}'[\lambda x.\, E^v[\mathcal{C}[v]]])\,\mathcal{C}[v]$$

Ici, le contexte de clôture $\mathcal{C}$ a été copié, ce qui provoquera le recalcul de ses termes liés si jamais ils sont utilisés au cours de la réduction. Ce phénomène n'est pas visible sur notre exemple $\Delta\,(I\,I)$, car le premier I ne duplique pas son argument. Au contraire, le terme suivant produirait une duplication de contexte inutile :

$$(\lambda x.\, (x\,I)\,x)\,((\lambda y.\, \lambda z.\, y)\,(I\,I))$$

car le terme à droite de l'application est déjà une valeur à clôture près, et il utilise un argument de sa clôture, le terme $(I)\,I$. Lors de la substitution, ce calcul est copié tel quel, ce qui fait que lors du calcul du corps de l'abstraction, chaque appel à x nécessitera de le recalculer. Le *call-by-need* d'Ariola et Felleisen évite cet écueil grâce à la règle Assoc.

On peut résoudre ce problème d'une façon simple et élégante, d'une manière similaire à ce que fait la règle Assoc : au cours de la substitution d'une valeur, on procède aussi à une extrusion de la clôture de cette valeur. Il n'y a pas besoin de procéder à un raffinement des contextes car tout y est déjà présent.

Le calcul résultant est précisé ci-dessous.

Définition 12. L'appel par valeur linéaire de tête avec partage est défini par l'unique réduction :

$$\mathcal{C}^{@}[(\mathcal{C}[\lambda x.\, \mathcal{C}^{\lambda}[E^v[x]]])\,\mathcal{C}'[v]] \to_{\lambda_{wlv}} \mathcal{C}^{@}[\mathcal{C}'[(\mathcal{C}[\lambda x.\, \mathcal{C}^{\lambda}[E^v[v]]])\,v]] \qquad \text{si } \mathcal{C}^{@} \ni \mathcal{C}^{\lambda}$$

avec les conventions habituelles pour éviter la capture de variables de $\mathcal{C}'$.

3.4. Comparaison avec les calculs en appel par nécessité

Assez remarquablement, le calcul ainsi obtenu est précisément le *call-by-need* de Felleisen et Chang, à l'exception qu'il utilise une substitution linéaire au lieu de la substitution destructive habituelle du λ-calcul. On rappelle ce calcul à la figure 3.4.

On se rend bien compte en particulier que les contextes de réponse A correspondent à nos contextes de clôture, et que les variantes internes et externes correspondent aux contextes lieurs et applicatifs respectivement.

De petites différences apparaissent, en particulier au niveau du comblement. Cette condition du calcul de Chang-Felleisen est exprimée comme une égalité, mais les deux formalismes sont équivalents, puisqu'il s'agit essentiellement de réarranger la manière dont on a découpé les contextes.

Valeurs	v	::=	$\lambda x.\, t$
Réponses	a	::=	$A[v]$
Contextes de réponses	A	::=	$[\cdot] \mid A_1[\lambda x.\, A_2]\, u$
Contextes de réponses internes	A^λ	::=	$[\cdot] \mid A[\lambda x.\, A^\lambda]$
Contextes de réponses externes	$A^@$	::=	$[\cdot] \mid A[A^@]\, u$
Contextes	E	::=	$[\cdot] \mid E\, u \mid A[E]$
			$\mid A^@[A[\lambda x.\, A^\lambda[E[x]]]\, E]$ avec $A^@[A^\lambda] \in A$

$$A^@[A_1[\lambda x.\, A^\lambda[E[x]]]\, A_2[v]] \longrightarrow A^@[A_1[A_2[(A^\lambda[E[x]])\{x := v\}]]] \quad \text{si } A^@[A^\lambda] \in A \qquad (\beta_{\text{cf}})$$

FIGURE 2 – *Call-by-need* de Chang-Felleisen

On peut exprimer β_{cf} dans notre formalisme de manière quasiment transparente.

Définition 13 (Chang-Felleisen *revisited*)**.** Le λ-calcul en appel par nécessité de Felleisen et Chang est décrit par la règle de réduction :

$$\mathcal{C}^@[(\mathcal{C}[\lambda x.\, \mathcal{C}^\lambda[E^v[x]]])\, \mathcal{C}'[v]] \to_{\beta_{\text{cfr}}} \mathcal{C}^@[\mathcal{C}[\mathcal{C}'[(\mathcal{C}^\lambda[E^v[v]])\{x := v\}]]] \qquad \text{si } \mathcal{C}^@ \ni \mathcal{C}^\lambda.$$

Remarque 5. On notera que le calcul de Chang et Felleisen emboîte les clôtures dans l'ordre opposé de notre calcul. Alors que l'utilisation d'une substitution non linéaire et destructrice permet ce choix, l'utilisation d'une substitution linéaire nous force à faire le choix présenté plus haut.

Morale 1. La réduction linéaire de tête faible par valeur avec partage de clôtures *est* un λ-calcul en appel par nécessité.

3.5. Vers la pleine paresse

Évaluation pleinement paresseuse. Les calculs considérés jusque là ne capturent pas ce que l'on appelle la pleine paresse. Par exemple, pour reprendre l'exemple initial, l'évaluation de $(I)\, I$ dans $(\Delta)(I)I$ est partagée dans les calculs considérés jusque là, mais elle n'est pas partagée dans $(\Delta)\lambda x.\, ((I)\, I)\, x$ (obtenu par η-expansion du premier terme).

L'interpréteur de Wadsworth va quant à lui partager le calcul de $(I)\, I$; ce niveau de partage des calculs est désigné sous l'appellation d'évaluation pleinement paresseuse (ou *fully lazy*).

Ariola et Felleisen [3] proposent un calcul pour la pleine paresse dans la dernière section de leur article sous le nom de calcul de Wadsworth :

Définition 14 (Sous-expressions libres)**.** Soit t une abstraction et u un sous-terme de t. On dira que u est une expression libre de t si aucune variable libre de u n'est liée dans t et si u est une application $(v)\, w$.

On dira qu'un expression libre de t, u, est une expression libre maximale si aucune expression libre de t ne contient strictement u.

Étant donnés des termes t et $\vec{u} \equiv u_1, \ldots u_n$, $\text{mfe}(t, \vec{u})$ signifiera que les u_i sont des expressions libres maximales de t. La métavariable V_{mfe} dénotera les valeurs ne contenant aucune expression libre (maximale).

Définition 15 (Calcul de Wadsworth)**.** Le calcul de Wadsworth est obtenu en modifiant le calcul d'Ariola et Felleisen (présenté en définition 1) par la décomposition de la règle DEREF en deux règles

$\textsc{Deref}_{\text{wad}}$ et $\textsc{Mfe}_{\text{wad}}$:

$$\begin{array}{lllll}
(\textsc{Deref}_{\text{wad}}) & (\lambda x.\, E[x])\, V_{\text{mfe}} & \to & (\lambda x.\, E[V_{\text{mfe}}])\, V_{\text{mfe}} \\
(\textsc{Mfe}_{\text{wad}}) & (\lambda x.\, E[x])\, \lambda y.\, C[\vec{t}] & \to & (\lambda x.\, E[x])\, (\lambda \vec{z} y.\, C[\vec{z}])\, \vec{t} \\
 & & & \text{si } \text{mfe}(\lambda y.\, C[\vec{t}], \vec{t}) \text{ et } \nexists u, \text{mfe}(\lambda y.\, C[\vec{z}], u) \\
(\textsc{Lift}) & (\lambda x.\, A)\, t\, u & \to & (\lambda x.\, (A)\, u)\, t \\
(\textsc{Assoc}) & (\lambda x.\, E[x])\, (\lambda y.\, A)\, t & \to & (\lambda y.\, (\lambda x.\, E[x])\, A)\, t
\end{array}$$

Le calcul proposé par Chang et Felleisen, de même que le calcul synthétisé dans les précédentes sections, ne capture pas la pleine paresse. Même s'ils effectuent bien du partage de clôtures, ces calculs ne vont pas partager les calculs qui pourraient être contenus dans une valeur.

Vers une évaluation pleinement paresseuse. Pourtant, la σ-équivalence nous apporte un éclairage intéressant sur les expressions libres. La règle suivante (avec les conditions usuelles sur les variables liées) :

$$\lambda x.\, (\lambda y.\, t)\, u \to (\lambda y.\, \lambda x.\, t)\, u$$

est une instance de la σ-équivalence de Regnier. Le contenu calculatoire de cette règle consiste à prendre une valeur contenant une clôture, à exposer cette clôture en la faisant commuter avec l'abstraction et ainsi à la rendre accessible au partage de clôtures.

Puisqu'on a l'hypothèse que u ne contient pas x comme variable libre, on voit que (dans le cas où u est une application), u est une sous-expression libre maximale du terme considéré et qu'on est tout près de la réduction mfe puisque celle-ci nous donnerait :

$$\lambda x.\, (\lambda y.\, t)\, u \to_{\textsc{Mfe}_{\text{wad}}} (\lambda z.\, \lambda x.\, (\lambda y.\, t)\, z)\, u$$

Il est donc tentant d'incorporer de la détection de sous-expression libre à base de σ-équivalence dans notre calcul provenant de la réduction linéaire de tête. Nous n'irons pas plus loin dans cette direction dans le présent article car le cadre que nous avons adopté jusque là, celui d'un calcul avec une unique réduction macroscopique se prête difficilement à mener cette idée jusqu'au bout.

En revanche, la dérivation d'un calcul en appel par nécessité depuis une version atomique de la réduction linéaire de tête devrait nous permettre d'être (presque) pleinement paresseux.

4. Conclusion

Dans cet article, nous avons dérivé de manière systématique un calcul en appel par nécessité à partir de la réduction linéaire de tête en trois étapes : restriction à une réduction faible, passage de valeurs, partage de clôtures. Au passage, nous avons reformulé la réduction linéaire de tête qui, dans la littérature, est présentée d'une manière relativement peu praticable.

Les ingrédients pour aboutir à ce résultat ont été :
— la clarification du fonctionnement de la réduction linéaire de tête,
— la mise en évidence de la notion de contexte de clôture et le fait de placer cette structure au centre de notre construction : toutes les constructions que nous donnons sont "closes" pour les contextes de clôtures,
— la prise en sérieux de la σ-équivalence.

Réduction macroscopique *vs.* **réduction microscopique.** Partant d'un calcul avec une réduction macroscopique, nous arrivons à un appel par nécessité macroscopique (c'est-à-dire définie par un unique axiome). Nous aurions pu prendre un autre chemin, que nous explorerons à l'avenir : on peut sans aucun problème intégrer les règles de la σ-équivalence dans le calcul de la réduction linéaire

de tête de manière à définir une réduction linéaire de tête progressive. Dans ce cas, nous conjecturons qu'on serait arrivé précisément sur le calcul d'Ariola et Felleisen. L'approche microscopique semble également prometteuse pour s'approcher de la pleine paresse.

Liens entre réduction linéaire de tête et appel par nécessité. Les liens entre réduction linéaire de tête et appel par nécessité sont frappants. Il est tout aussi frappant de voir qu'on ne trouve pas de lien dans la littérature entre appel par nécessité et réduction linéaire de tête. Par exemple dans l'introduction d'un article récent de Danvy et al. [10], un calcul en appel par nom avec let est présenté avec un objectif d'illustration. Ce calcul est essentiellement un calcul pour la réduction linéaire de tête.

Réseaux de preuve et call-by-need. À notre sens, cette connexion est potentiellement fructueuse : l'expression de l'appel par nécessité à partir de la réduction linéaire de tête va nous permettre de mener une analyse de l'appel par nécessité dans les réseaux de preuve, en allant possiblement jusqu'à l'analyse de la pleine paresse comme le suggère la section précédente. Les liens entre logique linéaire et appel par nécessité sont à creuser car ils vont bien au-delà de l'ébauche qu'on trouve dans [14].

Dans ce sens, il sera également utile de considérer les liens entre notre approche et le λ-calcul structurel d'Accattoli et Kesner [1].

Vers le contrôle. Une autre ligne de travail à venir consiste à considérer l'extension au contrôle. Dans des travaux précédents, le second auteur a déjà proposé une extension de l'appel par nécessité au contrôle [5, 2]. La méthodologie était différente même si, là encore, la méthode reposait sur la théorie de la preuve. Il s'agissait de partir de calculs construits sur le calcul des séquents pour les adapter à l'appel par nécessité. Dans cette conception, l'intégration du contrôle est transparente mais c'est la spécification de ce qu'est la paresse qui est loin d'être évidente et pas forcément canonique. (En particulier on observait dans ces travaux que différentes manières d'intégrer l'appel par nécessité et le contrôle résultaient en des calculs différents.) Ici, on est dans une configuration différente : la spécification de l'appel par nécessité est maintenant solidement justifié et il va falloir intégrer le contrôle.

Références

[1] B. Accattoli and D. Kesner. The structural *lambda*-calculus. In *Computer Science Logic, CSL*, volume 6247 of *Lecture Notes in Computer Science*, pages 381–395. Springer, 2010.

[2] Z. M. Ariola, P. Downen, H. Herbelin, K. Nakata, and A. Saurin. Classical call-by-need sequent calculi : The unity of semantic artifacts. In *Functional and Logic Programming Symposium, FLOPS 2012*, volume 7294 of *Lecture Notes in Computer Science*, pages 32–46. Springer, 2012.

[3] Z. M. Ariola and M. Felleisen. The call-by-need lambda calculus. *Journal of Functional Programming*, 7(3) :265–301, 1997.

[4] Z. M. Ariola, M. Felleisen, J. Maraist, M. Odersky, and P. Wadler. The call-by-need lambda calculus. In R. K. Cytron and P. Lee, editors, *Symposium on Principles of Programming Languages, POPL*, pages 233–246. ACM Press, 1995.

[5] Z. M. Ariola, H. Herbelin, and A. Saurin. Classical call-by-need and duality. In *Typed Lambda Calculi and Applications*, volume 6690 of *lncs*, 2011.

[6] S. Chang and M. Felleisen. The call-by-need lambda calculus, revisited. *European Symposium on Programming, ESOP*, abs/1201.3907, 2012.

[7] P.-L. Curien and H. Herbelin. The duality of computation. In *International Conference on Functional Programming, ICFP*, 2000.

[8] V. Danos, H. Herbelin, and L. Regnier. Game semantics & abstract machines. In *Logic in Computer Science, LICS*, pages 394–405, 1996.

[9] V. Danos and L. Regnier. Head linear reduction. Technical report, 2004.

[10] O. Danvy, K. Millikin, J. Munk, and I. Zerny. Defunctionalized interpreters for call-by-need evaluation. In *Functional and Logic Programming Symposium, FLOPS2010*, 2010.

[11] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50 :1–102, 1987.

[12] M. Hyland and L. Ong. On full abstraction for PCF. *Information and Computation*, 163(2) :285–408, Dec. 2000.

[13] J.-L. Krivine. A call-by-name lambda-calculus machine. *Higher-Order and Symbolic Computation*, 20(3) :199–207, 2007.

[14] J. Maraist, M. Odersky, D. N. Turner, and P. Wadler. Call-by-name, call-by-value, call-by-need and the linear lambda calculus. *Electronic Notes in Theoretical Computer Science*, 1 :370–392, 1995.

[15] J. Maraist, M. Odersky, and P. Wadler. The call-by-need λ-calculus. *Journal of Functional Programming*, 8(3) :275–317, 1998.

[16] L. Regnier. *Lambda-calcul et réseaux*. PhD thesis, Univ. Paris VII, 1992.

[17] L. Regnier. Une équivalence sur les lambda-termes. *Theoretical Computer Science*, 126 :281–292, 1994.

[18] C. P. Wadsworth. *Semantics and pragmatics of the lambda-calculus*. PhD thesis, Programming Research Group, Oxford University, 1971.